Compression of Image Clusters using Karhunen Loeve Transformations

Matthias Kramm

TU-München, Institute for Computer Science, Boltzmannstr. 3, 85748 Garching, Germany

ABSTRACT

This paper proposes to extend the Karhunen-Loeve compression algorithm to multiple images. The resulting algorithm is compared against single-image Karhunen Loeve as well as algorithms based on the Discrete Cosine Transformation (DCT).
Futhermore, various methods for obtaining compressable clusters from large image databases are evaluated.

Keywords: Karhunen Loeve Compression, KLT, Image Clustering, Image Similarity, Multi-Image Compression

1. INTRODUCTION

In large scale multimedia archive systems, it's often desirable to apply image compression not only to individual images, but also to groups of images, in order to gain better compression rates by exploiting inter-image redundancies, and mimimize the overhead of compression header information like coding trees and lookup tables.

Algorithms for compressing such image databases need first to divide the images into manageable blocks (clusters), and then apply a given compression algorithm (extended to a multi-image version) to each cluster.

In this paper, the Karhunen-Loeve image compression algorithm is analyzed for its multi-image compression and clustering capabilities.

The Karhunen-Loeve compression (a transformation based algorithm, henceforth abbreviated by KLT) is usually ineffective in practice for two-dimensional image compression, because a rather large transformation matrix has to be stored together with the compressed data (unlike JPEG, where the transformation matrix is hardcoded). For compression of clusters, however, the matrix has only to be stored once per cluster, so derivation of a dedicated matrix pays off.

Generalizing the matrix across more than one image will also cause the ompression rates for the individual images to decrease slightly as the KLT Matrix is now derived from a larger set of images- that is, the presence of additional statistical elements in the covariance matrix deterioates the compression gains for all of the images. As practical results show, however, this effect is neglectable, and the fact that almost always a transformation matrix better than DCT is used for compression without the tradeoff of having to store a different transformation matrix structures together with the data make the algorithm outperform its DCT variant.

2. PREVIOUS WORK

Surprisingly little previous work can be found about the area of generalized image cluster compression. A number of special cases are available, though, most notably the area of hyperspectral compression, which deals with the coding of a number (heavily correlated) images of the same dimensions. The algorithm for multispectral 3D image compression presented in [STR05] is refined and adapted in a number of other papers, for example [Lee99], and [DC04]. Lossless compression of correlated hyperspectral images is discussed e.g. in [WBS05]. [Kes04] also deals with distance metrics between spectra. Another special case for cluster compression is the area of digital video compression. E.g. [WLMC$^+$04] gives an overview over the current state-of-the-art. This area effectively deals with the same special case as hyperspectral compression, in that all consecutive images are

Further author information:
Matthias Kramm, eMail: kramm@in.tum.de, Telephone: +49 89 289-18680

usually highly correlated and of the same dimension. Example for work which deals with (multi-image) video compression using KLT are [FG02] and [FG04].

Concerning clustering algorithms, a good overview is given in [JD88] and [JMF99]. The clustering algorithms nCut, MST, SAHN and RACE are furthermore described in [SM00], [XOX01], [DE84] and [Run] , respectively.

3. TRADITIONAL KLT

Karhunen-Loeve image compression [Sal00] is a transformation-based method for compressing images, in which (unlike algorithms like JPEG, where the transformation is known beforehand) a special matrix is calculated for (or based on) the image data. This matrix has the property that it decorrelates the covariance matrix of the data, which in most cases leads to superior compression.

Given a number of quadratic images ψ_i of size $n \times n$, the KLT transformation is defined by

$$\phi_i(x, y) = \int K(x, y, x', y')\psi_i(x', y')dx'dy'$$

with K such that

$$\forall (x_1, y_1) \neq (x_2, y_2) : Cov_\phi((x_1, y_1), (x_2, y_2))) = \int \phi_i(x_1, y_1)\phi_i(x_2, y_2)di = 0$$

with the ψ_i being a decomposition of the original image into quadratic blocks.

In order to calculate K, an principal component analysis of

$$Cov_\psi = \int \psi_i(\cdot, \cdot)\psi_i(\cdot, \cdot)di$$

has to be performed, which requires $O(n^6)$ operations [GL96]. Futhermore, K needs to be stored together with the (compressed) ϕ_i, requiring $O(n^4)$ disk space. Considering the speed/space tradeoff involved, a reasonable choice for n is e.g. 8.

After transforming the source data, in order to compress the ϕ_i blocks, usually a quantisation step followed by run-length and Huffman or arithmetric encoding is used. (E.g. see [EHSMC92])

DCT based compression can be treated as a special case of this algorithm, in which the K function is hardcoded as

$$K_{DCT}(x, y, x', y') = C(x)C(y)cos(\frac{\pi x(2x' + 1)}{2n})cos(\frac{\pi y(2y' + 1)}{2n})$$

with $C(x) = \frac{1}{\sqrt{2}}$ for $x = 0$ and $C(x) = 1$ otherwise.

Reversely, this implies [CNT] that the Cov_ϕ diagonalized by K_{DCT} is a matrix $T + H$, with T being a Toeplitz and H a Hankel matrix.

4. CLUSTER KLT

When extending “the” KLT algorithm for cluster compression, the ψ_i blocks are derived from more than one image, which means the correlation function

$$Cov_\psi(x_1, y_1, x_2, y_2) = \int \psi_i(x_1, y_1)\psi_i(x_2, y_2)di$$

will not be for blocks of a single bitmap, but between all blocks of the image cluster to compress.

Depending on the images material combined into groups, the derived KLT matrix K will now decorrelate a correspondingly higher number of statistical sources (It can be shown [GS58] that given enough images, the KLT matrix will ultimatively converge against K_{DCT}).

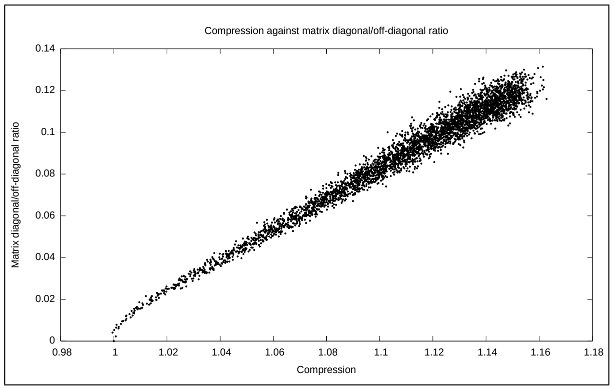

Figure 1. Estimation of compression rate by covariance matrix For a given image , the covariance matrix of the image blocks has been increasingly randomly pertubed, and then been used for compression. Both file size (vertical) as well as absolute diagonal/off-diagonal ratio has been measured. A linear model for this case is *file size = 2880.5 + 366.3 ratio*, with an standard error of ± 0.005741. (R^2 value of 0.9716).

The extended algorithm needs the transformation matrix to be stored only once per cluster. For every new image, the matrix has to be updated, however, and the existing images recompressed.

For the author's implementation, compression isn't done until the cluster reaches a reasonable size (for example, 16), and after that the matrix is considered as being "converged" and not updated any longer for new images. (Only then will the first 16 images be compressed using the derived matrix, and no recompression takes place anymore for the 17th, 18th, ... image)

For serial compression, every new image will be assigned to the cluster it best fits into.

In order to evaluate which cluster is optimal for an image, a number of methods can be used.

The most easy (and also the most precise) way to find out whether a given cluster compresses an image better than some other cluster is to compress the image with both clusters, and then compare the results (In the later chapters, this algorithm will be referenced as the "dummy compression" algorithm.)

If we're satisfied only with an estimate of the correct metric, however, some optimizations can be done.

The first improvement which can be made is to omit the the final compression steps. That is, after the transformation not a full compression run is done, but only a few statistics about the transformed block coefficients are calculated. This is very similar to the way most motion estimation algorithms in video compression work. Statistics which are usually generated in this area are, for example, creating sums of absolute total differences, taking the biggest coefficient, or adding the number of bits used. The number of papers is slim in this area, but looking at the source code of some standard video applications (like MPlayer (www.mplayerhq.hu/) reveals about a dozen possible algorithms.

The second refinement goes one step further, and omits the transformation itself, too. An estimate of how good the image will compress is therefore only derived from the correlation matrix of the image in question, and the KLT matrix which would be used for transformation. Notice that the correlation matrix for all images is always known, as it has to be calculated in order to derive the KLT transformation. The degree to which an image belongs to a given cluster can be evaluated by multiplying the correlation matrix with the transformation matrix. The optimal case would yield a diagonal matrix, while the worst case yields a full matrix. Intermediate cases can hence be classified by comparing the absolute sum of diagonal entries to the absolute sum on the diagonal (In the later chapters, this algorithm will be referenced as the "diagonal / off-diagonal" algorithm.) See figure 1 for details.

The presented algorithm (Figure 2) uses the latter method, deriving the estimated compression only from the product of the cluster's KLT matrix, and the covariance matrix of the image. Especially for big images and/or a big number of clusters, the resulting matrix operations require only a fraction of the time a dummy compression run would take.

```
add_new_image(image) {
    A $\Leftarrow$ covariance matrix of image
    Ci $\Leftarrow$ all products of A with cluster's klt matrix
    find Ci with best diagonal/off-diagonal ratio,
    C $\Leftarrow$ corresponding cluster
    if clustersize = 16:
        add image, compress cluster C
    if clustersize < 16:
        add image to cluster C
    if clustersize > 16:
        add compressed image to cluster C
}
```

Figure 2. *C*lustering algorithm

The KLT compression itself is done via a variant of the h.263 video compression algorithm. In particular, the resulting coefficients of the matrix transformation are compressed by a quantization step (with quantisation factor q, and coefficient c)

$$c_q = int(c/q)$$

which is followed by run length encoding (RLE). The resulting run-lengths and non-zero coefficients are then encoded using hardcoded Huffman coefficients. [IT98]

5. PARTITIONING OF IMAGES

If no serial compression of the images is required, and the whole image database is known beforehand (e.g. when creating an image CDROM or DVD), more sophisticated clustering algorithms can be used.

Clustering algorithms can be seperated into "hierarchical" and "non-hierachical" [JD88]. "Hierarchical" means that the algorithm will always deal with clusters, not individual nodes when forming the resulting clusters, while a "non-hierachical" algorithm will only deal with nodes.

Using the "dummy compression" or "diagonal/off-diagonal" image comparison functions from the previous chapter (figure 1) , we can compare both images with images as well as images with clusters and clusters with clusters. Therefore, we can use both hierarchical as well as non-hierachical algorithms.

Notice that in order to compare a cluster with another cluster, the accumulative covariance matrix Cov_ψ needs to be stored together with the cluster. This takes only $O(n)$ space, however, because we can store this as $D = K^T Cov_\psi K$, i.e. the eigenvalues of Cov_ψ. An overview over all metric functions is given in the following table:

A	B	Compression metric	Matrix metric
Image	Image	Compress A,B with accumulative $K(Cov_{\psi_{AB}})$	-
Image	Cluster	Compress image with K_B	diagonal/off-diagonal ratio of $K_B Cov_{\psi_A}$
Cluster	Cluster	Compress all images in both clusters with $K(Cov_{\psi_{AB}})$	diagonal/off-diagonal ratio of $K_B Cov_{\psi_A} + K_A Cov_{\psi_B}$

Possible choices for the hierarchical clustering algorithm are RACE, SAHN, nCut, or MST (Minimum spanning tree clustering). Furthermore, we can just iterate over the images and assign them to the cluster which is momentarily best (Greedy), or build arbitrary clusters (Random). The Greedy algorithm can be slightly refined by ensuring that all clusters are kept above a minimal size, in order to prevent against clusters which contain

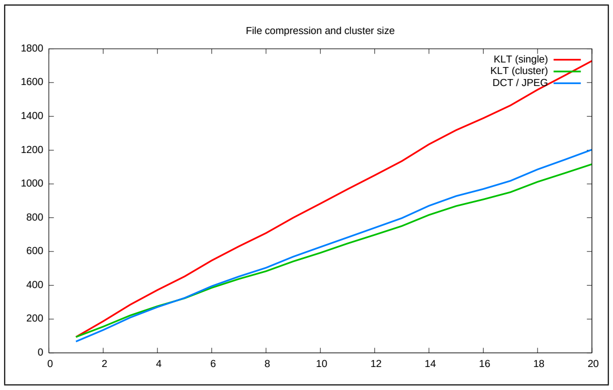

Figure 3. *Comparision of cluster KLT compression with KLT and DCT*

only one or two nodes (Greedy-MIN). While SAHN is the only "real" hierarchical algorithm, RACE and Greedy will also need to do cluster-cluster and cluster-node comparisons.

A comparison of the clustering results of 1000 images is given in the following table:

Algorithm	Clusters	Compressed Size	
SAHN	6	8061436	46 / 36 / 14 / 14 / 12 / 6
Greedy-MIN	6	8082217	77 / 11 / 11 / 11 / 9 / 9
MST	6	8097069	123 / 1 / 1 / 1 / 1 / 1
Greedy	6	8106955	111 / 10 / 2 / 2 / 2 / 1
RACE	6	8141613	27 / 24 / 23 / 19 / 18 / 17
Random	6	8175611	28 / 23 / 21 / 20 / 19 /17

It's visible that MST (which is graph-theoretically the best algorithm) tends to cut out only single images in order to form the 6 clusters given as input here. It still outperforms greedy however, which shows a similar "symptom"- once some cluster has reached a given size, all new images get directed to that cluster (because its KLT matrix is the most "generalized" due to having received more input data). The enhanced Greedy algorithm Greedy-MIN rectifies that drawback, and hence beats MST. RACE (even though it creates more balanced clusters) gives the worst result (other than poor random), which is suprising because it also seems to be the most expensive algorithm, concerning computing time (depending on the number of iterations).

In the author's (serial) implementation, the Greedy-MIN algorithm was used.

For a non-serial algorithm, SAHN is preferable as clustering algorithm, as it provides a reasonable compression / computing-time ratio, and the best results.

6. COMPRESSION GAINS

The proposed algorithm has been compared with both a JPEG-like DCT transformation algorithm (the algorithm used for compressing intraframes in h.263 video) and single image KLT compression (where a matrix is derived for every single image). As illustrated by the graph in figure 3, DCT compression is superior for a very small number of images, as there's no transformation matrix to be stored. For a larger number of images, KLT compression which stores the transformation matrix only once per cluster pays off.

A notable result is also that even for a cluster of 4000 images used in one of the longer test runs, which resulted in around 30,000,000 individual 8x8 blocks, the KLT tranformation matrix still performs better than a normal DCT matrix. That is, even though the KLT converges against the DCT in practice as well as theory [GS58], the remaining statistical influence of the image source material is still beneficial after several millions of 8x8 blocks. In figure 5, the eigenimages for the 4000 photo testset is depicted. This is mostly just a permutation of the DCT matrix, with a small layer of noise not present in "clean" DCT. More interesting are the eigenimages of the "thesis" sample set, figure 6, because here some principal components are different (more "edgy") from the standard Fourier matrix. For the sake of comparison, the original DCT is depicted in figure 4.

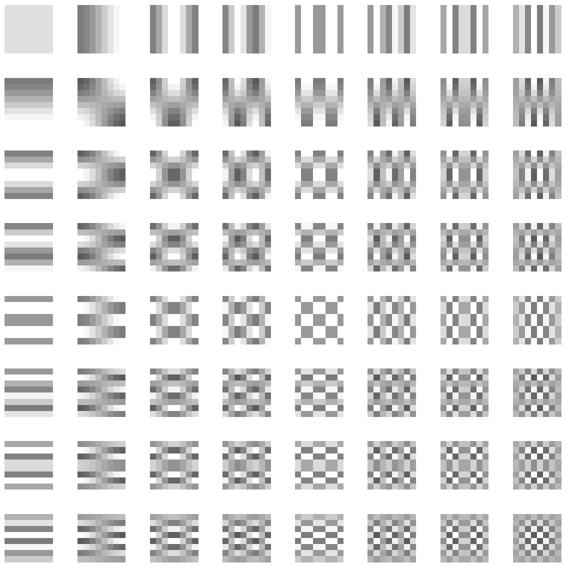

Figure 4. *DCT base images*

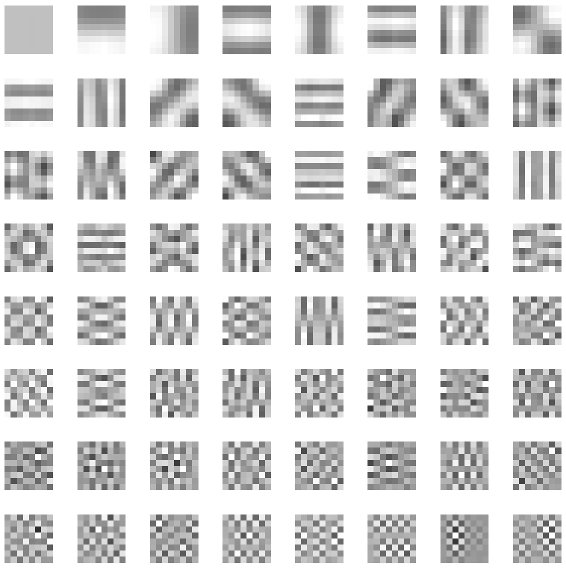

Figure 5. *Base images of a KLT transformation for 4000 photo images*

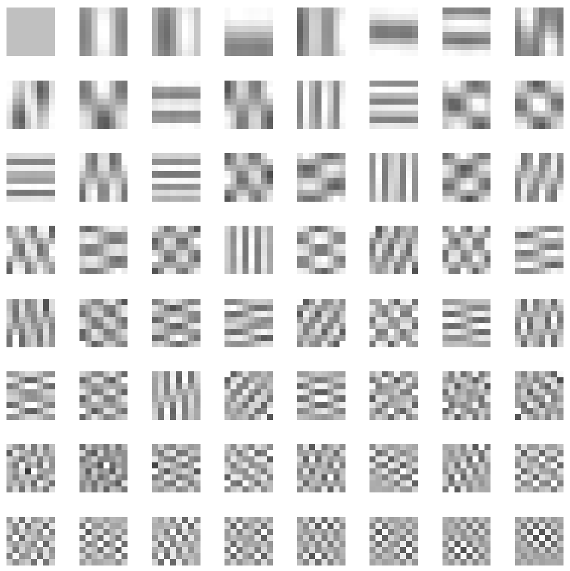

Figure 6. *Base images of a KLT transformation for 4000 text page images*

7. SUMMARY

In this paper, an extension of the KLT-transformation compression algorithm for use in image cluster compression has been evaluated.

The proposed extended algorithm in its serial version uses a greedy algorithm based on the distance between the unit matrix and the product of the image's covariance matrix and the cluster's transformation matrix in order to match images against clusters.

The actual compression is then done via a KLT matrix derived from the first 16 images of a cluster, and subsequent RLE and Huffman steps. Compared to DCT compression as well as single-image KLT compression algorithms, the proposed method performs better for a reasonably high number of images with only a low computational overhead.

Explicit SAHN clustering can be used to refine the algorithm further, should the image data be known beforehand in its entirety.

REFERENCES

1. J. Saghri, A. Tescher, and J. Reagan, "Practical transform coding of multispectral imagery," *Signal Processing Magazine, IEEE* , pp. 32–43, 2005.
2. J. Lee, "Optimized quadtree for karhunen-loeve transform in multispectral image coding," *Image Processing, IEEE Transactions on* **8**, pp. 453–461, 1999.
3. Q. Du and C.-I. Chang, "Linear mixture analysis-based compression for hyperspectral image analysis," *Geoscience and Remote Sensing, IEEE Transactions on* **42**, pp. 875–891, 2004.
4. H. Wang, D. S. Babacan, and K. Sayood, "Lossless hyperspectral image compression using context-based conditional averages," *Data Compression Conference, 2005. Proceedings. DCC 2005* , pp. 418–426, 2005.
5. N. Keshava, "Distance metrics and band selection in hyperspectral processing with applications to material identification and spectral libraries," *Geoscience and Remote Sensing, IEEE Transactions on* **42**, pp. 1552–1565, 2004.
6. Z. Wen, Z. Liu, . M. Cohen, J. Li, K. Zheng, and . T. Huang, "Low bit-rate video streaming for face-to-face teleconference," *Multimedia and Expo, 2004. ICME '04. 2004 IEEE International Conference on* **3**, pp. 1631–1634 Vol.3, 2004.
7. M. Flierl and B. Girod, "Video coding with motion compensation for groups of pictures," *Image Processing. 2002. Proceedings. 2002 International Conference on* **1**, pp. I–69–I–72 vol.1, 2002.
8. M. Flierl and B. Girod, "Video coding with motion-compensated lifted wavelet transforms," *Special Issue on Subband/Wavelet Interframe Video Coding* **19 issue 7**, pp. 561–57, 2004.
9. A. Jain and R. Dubes, *Algorithms for clustering Data*, Prentice Hall, 1988.
10. A. Jain, M. Murty, and P. Flynn, "Data clustering: A review," *ACM Computing Surveys* **31**, 1999.
11. J. Shi and J. Malik, "Normalized cuts and image segmentation," *IEEE Transactions on Pattern Analysis and Machine Intelligence* **22**(8), pp. 888–905, 2000.
12. Y. Xu, V. Olman, and D. Xu, "Minimum Spanning Trees for Gene Expression Data Clustering," *Genome Informatics* **12**, pp. 24–33, 2001.
13. W. Day and H. Edelsbrunner, "Efficient algorithms for agglomerative hierarchical clustering methods," *Journal of Classification* **1**(1), pp. 7–24, 1984.
14. T. A. Runkler, *Information Mining*, Vieweg AG.
15. D. Salomon, *Data compression, the complete reference*, 2000.
16. G. H. Golub and C. F. V. Loan, *Matrix Computations*, Johns Hopkins University Press, 1996.
17. B. Epstein, R. Hingorani, J. Shapiro, and D. S. M. Czigler, "Multispectral klt-wavelet data compression for landsat thematicmapper images," *dcc* , pp. 200–208, 1992.
18. R. H. Chan, M. K. Ng, and W.-C. Tang, "Deblurring models with neumann boundary conditions."
19. U. Grenander and G. Szego, *Toepliz forms and their applications*, University of California Press Berkeley, 1958.
20. ITU-T, "h.263 - video coding for low bit rate communication." CCITT recommendation, 1998.